cepton_sdk Documentation

Cepton Technologies

Oct 31, 2019

CONTENTS

1 Overview 1

2 Building 5

3 Samples 7

4 Internal 23

5 Legal 25

6 Tools 27

7 API Reference 33

Index 55

CHAPTER

ONE

OVERVIEW

The Cepton SDK provides the following features:

- **Networking**: Listen for sensor packets.
- **Capture Replay**: Read sensor packets from a PCAP file.
- **Parsing**: Parse sensor packets.
- **Calibration**: Apply sensor calibration.
- **Frame Accumulation**: Accumulate sensor points and detect frames.

Note: Currently, the Cepton LiDAR packet formats are under active development, and are not publicly available. The SDK is required for **Parsing** and **Calibration**. All other SDK features are optional, and can be done manually by the user.

1.1 Getting Started

Below is a very simple SDK usage example. For more complete examples, see *Samples*.

```
#include <cepton_sdk_api.hpp>

int main(int argc, char **argv) {
  // Initialize SDK with default options
  CEPTON_CHECK_ERROR(
      cepton_sdk::api::initialize(cepton_sdk::create_options(), "", true));

  // Get all sensors
  for (int i = 0; i < cepton_sdk::get_n_sensors(); ++i) {
    cepton_sdk::SensorInformation sensor_info;
    CEPTON_CHECK_ERROR(
        cepton_sdk::get_sensor_information_by_index(i, sensor_info));
  }

  // Listen for points
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  CEPTON_CHECK_ERROR(
      callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
                          const cepton_sdk::SensorImagePoint *c_image_points) {
        // Get sensor info
        cepton_sdk::SensorInformation sensor_info;
```

(continues on next page)

(continued from previous page)

```
        CEPTON_CHECK_ERROR(
            cepton_sdk::get_sensor_information(handle, sensor_info));

        // Convert points
        std::vector<cepton_sdk::util::SensorPoint> points(n_points);
        for (int i = 0; i < n_points; ++i) {
          cepton_sdk::util::convert_sensor_image_point_to_point(
              c_image_points[i], points[i]);
        }
      }));

  // Sleep or do other work...

  // Deinitialize SDK
  cepton_sdk::deinitialize().ignore();
}
```

For prototyping, it is recommended to use the high level C++ API. The C++ API also acts as reference code for C API usage. Any C++ functions that directly wrap C functions are not documented; refer to the C function documentation.

The general C SDK workflow is as follows:

1. Initialize SDK (*`cepton_sdk_initialize()`*). See *Setup*.
2. Register point listener callback function (*`cepton_sdk_listen_image_frames()`*). See *Points*.
3. Wait for sensor calibration/information packets, then query sensors. See *Sensors*.
4. Sleep or run replay. Callbacks will occur asynchronously.
5. Deinitialize SDK.

1.2 Packets

The SDK passively listens for sensor UDP packets. There are 2 types of sensor packets:

- **Calibration/Information**: Contains sensor calibration, statistics, and other information. Published at `~1Hz`.
- **Points**: Contains `~100` measurements. Published at `~1000Hz`.

1.3 Errors

Many SDK functions return *`CeptonSensorErrorCode`*. If this is not *`CEPTON_SUCCESS`*, then the user must call *`cepton_sdk_get_error()`*, otherwise the SDK will complain that the error was not checked.

All sensor errors will be returned via `FpCeptonSensorErrorCallback`, which is registered in *`cepton_sdk_initialize()`*.

1.4 Timestamps

All `int64` timestamps are microseconds since the Unix epoch (UTC). All `float` times (measurement period, replay time, frame length, etc.) are time differences measured in seconds. Point timestamps are based on one of the following sources (the first valid source is used):

1. GPS (NMEA + PPS)
2. PTP
3. Host PC

1.5 Sensor Fusion

See *Process Single*, *Process Multi*.

1.6 Multiple Returns

To enable multiple returns, pass the `CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS` flag during initialization.

The returns are as follows:

1. Strongest signal.
2. Furthest signal, if it is not the strongest. Otherwise, the second strongest signal.

CHAPTER

TWO

BUILDING

2.1 CMake

The simplest way to include the SDK is as a CMake subdirectory:

```
add_subdirectory(<cepton_sdk_source_dir>)

...

target_link_libraries(<target> cepton_sdk::cepton_sdk)
```

See *CMake*.

2.2 Manually

It is also possible to manually link to the SDK as follows:

1. Add the `cepton_sdk_redist/include` path.
2. Link to the correct library binary in `cepton_sdk_redist/lib` or `cepton_sdk_redist/bin`.
3. If statically linking, define `CEPTON_SDK_STATIC`.

2.3 Windows

Note: It is not possible to statically link in debug mode, since the library does not ship with debug symbols.

CHAPTER

THREE

SAMPLES

3.1 Building

3.1.1 Unix

```
cd cepton_sdk_redist/samples
mkdir build
cd build
cmake ..
make
```

3.1.2 Windows

The following commands are for a UNIX command line (e.g. Git Bash).

```
cd cepton_sdk_redist/samples
mkdir build
cd build
# You may need to modify the `Visual Studio` version number.
cmake -G "Visual Studio 16 2019" ..
```

To build from the command line, run

```
cmake --build . --config Release
```

To build in Visual Studio

- Click `File -> Open -> Project/Solution`. Select `cepton_sdk_redist/samples/build/cepton_sdk_samples.sln`.
- Build the project.

3.2 Basic

Start with *Basic* or *C Basic*.

3.2.1 CMake

Listing 1: samples/CMakeLists.txt

```
#[[
CMake file for building samples.
]]
cmake_minimum_required(VERSION 3.1)

set(CEPTON_SDK_SAMPLES_SOURCE_DIR "${CMAKE_CURRENT_LIST_DIR}")
get_filename_component(CEPTON_SDK_SOURCE_DIR
                       "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/../" ABSOLUTE)
list(APPEND CMAKE_MODULE_PATH "${CEPTON_SDK_SOURCE_DIR}/cmake")

include("CeptonVersion")
project(
  cepton_sdk_samples
  VERSION ${CEPTON_VERSION}
  LANGUAGES C CXX)

include("CeptonCommon")

if(GCC OR CLANG)
  add_flags(-Wall)
  add_linker_flags(-pthread)
endif()

# cepton_sdk
add_subdirectory("${CEPTON_SDK_SOURCE_DIR}"
                 "${PROJECT_BINARY_DIR}/third_party/cepton_sdk")

set(CEPTON_SDK_SAMPLE_SOURCES
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/frame_accumulator.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/frame_detector.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/process_multi.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/process_single.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/stray_filter.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/basic.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/callback.cpp"
    "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/error.cpp")
foreach(path ${CEPTON_SDK_SAMPLE_SOURCES})
  get_filename_component(name "${path}" NAME_WE)
  add_executable(cepton_sdk_sample_${name} "${path}")
  target_include_directories(cepton_sdk_sample_${name}
                             PRIVATE "${CEPTON_SDK_SAMPLES_SOURCE_DIR}")
  target_link_libraries(cepton_sdk_sample_${name} cepton_sdk::cepton_sdk)
endforeach()
```

3.2.2 Basic

Listing 2: samples/basic.cpp

```
/**
 * Sample code for general SDK usage.
 */
#include <cepton_sdk_api.hpp>

```

(continues on next page)

(continued from previous page)

```cpp
#include "common.hpp"

/// Sample points callback.
class FramesListener {
 public:
  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
                      const cepton_sdk::SensorImagePoint *c_image_points) {
    // Get sensor info
    cepton_sdk::SensorInformation sensor_info;
    CEPTON_CHECK_ERROR(cepton_sdk::get_sensor_information(handle, sensor_info));

    // Convert points
    static thread_local std::vector<cepton_sdk::util::SensorPoint> points;
    points.resize(n_points);
    for (int i = 0; i < (int)n_points; ++i) {
      cepton_sdk::util::convert_sensor_image_point_to_point(c_image_points[i],
                                                            points[i]);
    }

    // Print
    std::printf("Received %i points from sensor %i\n", (int)n_points,
                (int)sensor_info.serial_number);
  }
};

int main(int argc, char **argv) {
  // Parse arguments
  check_help(argc, argv, "cepton_sdk_sample_basic [capture_path]");
  std::string capture_path;
  if (argc >= 2) capture_path = argv[1];

  std::printf("Press Ctrl+C to stop\n");

  // Initialize SDK
  auto options = cepton_sdk::create_options();

  // By default, return points every packet.

  // Uncomment to return points every frame.
  options.frame.mode = CEPTON_SDK_FRAME_COVER;

  // Uncomment to return points at fixed time interval.
  // options.frame.mode = CEPTON_SDK_FRAME_TIMED;
  // options.frame.length = 0.1f;

  // Wait short duration for sensors to connect.
  const bool enable_wait = true;

  std::printf("Initializing...\n");
  CEPTON_CHECK_ERROR(
      cepton_sdk::api::initialize(options, capture_path, enable_wait));

  // Get all sensors
  const int n_sensors = (int)cepton_sdk::get_n_sensors();
  for (int i = 0; i < n_sensors; ++i) {
    cepton_sdk::SensorInformation sensor_info;
    CEPTON_CHECK_ERROR(
```

(continues on next page)

(continued from previous page)

```
        cepton_sdk::get_sensor_information_by_index(i, sensor_info));
    std::printf("Sensor: %i\n", (int)sensor_info.serial_number);
  }

  // Listen for points
  std::printf("Listening for points...\n");
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  FramesListener frames_listener;
  CEPTON_CHECK_ERROR(
      callback.listen(&frames_listener, &FramesListener::on_image_frame));

  // Run (sleep or run replay)
  CEPTON_CHECK_ERROR(cepton_sdk::api::wait(1.0));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.2.3 C Basic

Listing 3: samples/c_basic.c

```
/**
 * Sample code for general C SDK usage.
 */
#include <stdio.h>
#include <time.h>

#include <cepton_sdk.h>

void check_sdk_error() {
  const char *error_msg;
  const auto error_code = cepton_sdk_get_error(&error_msg);
  printf("%s: %s\n", cepton_get_error_code_name(error_code), error_msg);
  exit(1);
}

int n_frames = 0;

void image_frame_callback(CeptonSensorHandle handle, size_t n_points,
                          const struct CeptonSensorImagePoint *c_points,
                          void *user_data) {
  ++n_frames;

  // Get sensor info
  struct CeptonSensorInformation sensor_info;
  cepton_sdk_get_sensor_information(handle, &sensor_info);
  check_sdk_error();
}

int main() {
  // Initialize
  struct CeptonSDKOptions options = cepton_sdk_create_options();
  options.frame.mode = CEPTON_SDK_FRAME_COVER;
```

(continues on next page)

(continued from previous page)

```
  cepton_sdk_initialize(CEPTON_SDK_VERSION, &options, NULL, NULL);
  check_sdk_error();

  // Wait for sensor
  const int n_sensors = (int)cepton_sdk_get_n_sensors();
  while (!cepton_sdk_get_n_sensors())
    ;
  for (int i = 0; i < n_sensors; ++i) {
    struct CeptonSensorInformation sensor_info;
    cepton_sdk_get_sensor_information_by_index(0, &sensor_info);
    check_sdk_error();
    printf("Sensor: %i\n", (int)sensor_info.serial_number);
  }

  // Listen for frames
  cepton_sdk_listen_image_frames(image_frame_callback, NULL);
  check_sdk_error();

  // Sleep
  while (n_frames < 10)
    ;

  // Deinitialize
  cepton_sdk_deinitialize();
  check_sdk_error();
}
```

3.2.4 Callback

Listing 4: samples/callback.cpp

```
/**
 * Sample code for callback usage.
 */
#include <cepton_sdk_api.hpp>

// Sample global callback.
void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
                    const cepton_sdk::SensorImagePoint *c_image_points) {
  // Handle frame...
}

// Sample member callback.
class FramesListener {
 public:
  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
                      const cepton_sdk::SensorImagePoint *c_image_points) {
    // Handle frame...
  }
};

int main(int argc, char **argv) {
  // Initialize
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize());
  cepton_sdk::api::SensorImageFrameCallback callback;
```

(continues on next page)

(continued from previous page)

```
  CEPTON_CHECK_ERROR(callback.initialize());

  // Listen lambda
  CEPTON_CHECK_ERROR(
      callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
                         const cepton_sdk::SensorImagePoint *c_image_points) {
        //  Handle frame...
      }));

  // Listen global function
  CEPTON_CHECK_ERROR(callback.listen(on_image_frame));

  // Listen member function
  FramesListener frames_listener;
  CEPTON_CHECK_ERROR(
      callback.listen(&frames_listener, &FramesListener::on_image_frame));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.2.5 Error

Listing 5: samples/error.cpp

```
/**
 * Sample code for error callback usage.
 */
#include <cepton_sdk_api.hpp>

int main(int argc, char** argv) {
  // Initialize
  cepton_sdk::api::SensorErrorCallback error_callback;
  CEPTON_CHECK_ERROR(
      error_callback.listen([&](cepton_sdk::SensorHandle handle,
                                const cepton_sdk::SensorError& error) {
        // Handle error...
      }));
  CEPTON_CHECK_ERROR(cepton_sdk::initialize(
      CEPTON_SDK_VERSION, cepton_sdk::create_options(),
      error_callback.global_on_callback, &error_callback));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3 Advanced

3.3.1 Frame Detector

Listing 6: samples/advanced/frame_detector.cpp

```cpp
/**
 * Sample code for custom frame detection.
 */
#include <cepton_sdk_api.hpp>

int main(int argc, char **argv) {
  std::string capture_path;
  if (argc >= 2) capture_path = argv[1];

  // Initialize
  auto options = cepton_sdk::create_options();
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());

  // Get sensor
  while (cepton_sdk::get_n_sensors() == 0)
    CEPTON_CHECK_ERROR(cepton_sdk::api::wait(0.1f));
  cepton_sdk::SensorInformation sensor_info;
  CEPTON_CHECK_ERROR(
      cepton_sdk::get_sensor_information_by_index(0, sensor_info));

  // Create detector
  cepton_sdk::util::FrameDetector<> detector(sensor_info);
  auto frame_options = cepton_sdk::create_frame_options();
  frame_options.mode = CEPTON_SDK_FRAME_COVER;
  CEPTON_CHECK_ERROR(detector.set_options(frame_options));
  const int stride = sensor_info.segment_count * sensor_info.return_count;
  CEPTON_CHECK_ERROR(callback.listen(
      [&](cepton_sdk::SensorHandle handle, std::size_t n_points,
          const cepton_sdk::SensorImagePoint *const c_image_points) {
        if (handle != sensor_info.handle) return;

        for (int i = 0; i < (int)n_points; i += stride) {
          auto &image_point = c_image_points[i];
          if (detector.update(image_point)) {
            auto &result = detector.previous_result();
            // `detector.period()` Frame period [seconds].
            // `result.timestamp` Frame timestamp [microseconds].

            // Handle frame...
          }
        }
      }));

  // Run
  CEPTON_CHECK_ERROR(cepton_sdk::api::wait(1.0f));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.2 Frame Accumulator

Listing 7: samples/advanced/frame_accumulator.cpp

```cpp
/**
 * Sample code for custom frame accumulation.
 */
#include <cepton_sdk_api.hpp>

int main(int argc, char **argv) {
  std::string capture_path;
  if (argc >= 2) capture_path = argv[1];

  auto frame_options = cepton_sdk::create_frame_options();

  // Uncomment to return points every frame.
  frame_options.mode = CEPTON_SDK_FRAME_COVER;

  // Uncomment to return points at fixed time interval.
  // frame_options.mode = CEPTON_SDK_FRAME_TIMED;
  // frame_options.length = 0.1f;

  // Initialize
  auto options = cepton_sdk::create_options();
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());

  // Get sensor
  while (cepton_sdk::get_n_sensors() == 0)
    CEPTON_CHECK_ERROR(cepton_sdk::api::wait(0.1f));
  cepton_sdk::SensorInformation sensor_info;
  CEPTON_CHECK_ERROR(
      cepton_sdk::get_sensor_information_by_index(0, sensor_info));

  // Create accumulator
  cepton_sdk::util::FrameAccumulator accumulator(sensor_info);
  CEPTON_CHECK_ERROR(accumulator.set_options(frame_options));
  CEPTON_CHECK_ERROR(callback.listen(
      [&](cepton_sdk::SensorHandle handle, std::size_t n_points,
          const cepton_sdk::SensorImagePoint *const c_image_points) {
        if (handle != sensor_info.handle) return;
        accumulator.add_points(n_points, c_image_points);
      }));

  // Listen
  CEPTON_CHECK_ERROR(accumulator.callback.listen(
      [&](std::size_t n_points,
          const cepton_sdk::SensorImagePoint *const c_image_points) {
        // Handle frame...
      }));

  // Run
  CEPTON_CHECK_ERROR(cepton_sdk::api::wait(1.0f));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.3 Network

Listing 8: samples/advanced/network.cpp

```cpp
/**
 * Sample code for custom networking.
 */
#include <asio.hpp>

#include <cepton_sdk_api.hpp>

using asio::ip::udp;

class SocketListener {
 public:
  SocketListener() : m_socket(m_io_service, udp::v4()) {
    m_socket.set_option(asio::socket_base::reuse_address(true));
    m_socket.bind(udp::endpoint(udp::v4(), 8808));
  }

  void run() {
    listen();
    m_io_service.run_for(std::chrono::seconds(5));
  }

  void listen() {
    m_socket.async_receive_from(
        asio::buffer(m_buffer), m_end_point,
        [this](const asio::error_code& error, std::size_t buffer_size) {
          if (buffer_size == 0) return;
          if (error == asio::error::operation_aborted) return;
          const CeptonSensorHandle handle =
              m_end_point.address().to_v4().to_ulong();
          // For more accurate timestamps, a separate network receive thread
          // should be
          // used.
          const int64_t timestamp = cepton_sdk::util::get_timestamp_usec();
          CEPTON_CHECK_ERROR(cepton_sdk::mock_network_receive(
              handle, timestamp, m_buffer.data(), buffer_size));
          listen();
        });
  }

 private:
  asio::io_context m_io_service;
  udp::socket m_socket;
  udp::endpoint m_end_point;
  std::array<uint8_t, 4096> m_buffer;
};

int main() {
  // Initialize
  auto options = cepton_sdk::create_options();
  options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
  options.frame.mode = CEPTON_SDK_FRAME_COVER;
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));

```

(continues on next page)

(continued from previous page)

```
  // Listen for points
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  CEPTON_CHECK_ERROR(
      callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
                         const cepton_sdk::SensorImagePoint* c_image_points) {
        std::printf("Received %i points from sensor %lli\n", (int)n_points,
                    (long long)handle);
      }));

  SocketListener listener;
  listener.run();

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.4 Organize Points

Listing 9: samples/advanced/organize_points.cpp

```
/*********************************************************************
 **                                                                 **
 **  Copyright(C) 2019 Cepton Technologies. All Rights Reserved.    **
 **  Contact: https://www.cepton.com                                **
 **                                                                 **
 **  Sample code which opens a cepton sensor pcap file, organizes   **
 **  the points and continuously saves the most recent organized    **
 **  points to a frame to a cvs file "organized_cloud.cvs           **
 *********************************************************************/

#include <cepton_sdk_util.hpp>
#include <cepton_sdk/capture.hpp>
#include <cepton_sdk_api.hpp>

using namespace cepton_sdk::util;

int main(int argc, char** argv) {
  if (argc < 2) return -1;
  const std::string capture_path = argv[1];

  // Initialize sdk
  auto options = cepton_sdk::create_options();
  options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
  options.frame.mode = CEPTON_SDK_FRAME_COVER;
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));

  cepton_sdk::SensorInformation sensor_info;

  OrganizedCloud organized_cloud;
  std::ofstream os;

  cepton_sdk::Capture m_capture;
  CEPTON_CHECK_ERROR(m_capture.open_for_read(capture_path));

```

(continues on next page)

(continued from previous page)

```
  CEPTON_CHECK_ERROR(
      cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
                                   CEPTON_SDK_CONTROL_DISABLE_NETWORK));
  CEPTON_CHECK_ERROR(cepton_sdk_clear());

  // Listen for points
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
                      const cepton_sdk::SensorImagePoint* c_image_points) {

    cepton_sdk::get_sensor_information(handle,sensor_info);

    std::printf("Received %i points from sensor %lli\n", static_cast<int>(n_points),
                static_cast<long long>(handle));

    Organizer organizer(sensor_info);

    organizer.organize_points(n_points,
                              sensor_info.return_count,
                              c_image_points,
                              organized_cloud);

    os.open("organize_cloud.csv");
    for (const auto& point : organized_cloud.points)
    {
      if (point.valid)
      {
        float x = 0;
        float y = 0;
        float z = 0;
        cepton_sdk::util::convert_image_point_to_point(
            point.image_x, point.image_z, point.distance, x,
            y, z);

        os << x << "," << y << "," << z << "\n";
      }
    }
    os.close();
  });

  while (true) {
    cepton_sdk::Capture::PacketHeader header;
    const uint8_t* data;
    CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));

    const cepton_sdk::SensorHandle handle =
        static_cast<cepton_sdk::SensorHandle>(header.ip_v4) |
        CEPTON_SENSOR_HANDLE_FLAG_MOCK;
    CEPTON_CHECK_ERROR(cepton_sdk_mock_network_receive(
        handle, header.timestamp, data, static_cast<size_t>(header.data_size)));
  }
}
```

3.3.5 Process Multi

Listing 10: samples/advanced/process_multi.cpp

```cpp
/**
 * Sample code for processing offline data from multiple sensors.
 */
#include <cepton_sdk_api.hpp>

#include "common.hpp"

struct Frame {
  int64_t timestamp;
  std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
      image_points_dict;
};

class FrameAccumulator {
 public:
  void on_image_frame(
      cepton_sdk::SensorHandle handle, std::size_t n_points,
      const cepton_sdk::SensorImagePoint* const c_image_points) {
    cepton_sdk::util::LockGuard lock(m_mutex);

    // Add points to buffer
    auto& image_points = m_image_points_dict[handle];
    image_points.insert(image_points.end(), c_image_points,
                        c_image_points + n_points);

    check_and_publish();
  }

 private:
  void check_and_publish() {
    // Check if frame done
    const auto timestamp = cepton_sdk::api::get_time();
    if ((timestamp - m_timestamp) < int64_t(m_frame_length * 1e6f)) return;
    m_timestamp = timestamp;

    // Add frame to queue
    auto frame = std::make_shared<Frame>();
    frame->timestamp = timestamp;
    frame->image_points_dict = m_image_points_dict;
    m_image_points_dict.clear();
    queue.push(frame);
  }

 public:
  cepton_sdk::util::SingleConsumerQueue<Frame> queue;

 private:
  std::timed_mutex m_mutex;
  float m_frame_length = 0.1f;
  int64_t m_timestamp = 0;
  std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
      m_image_points_dict;
};

int main(int argc, char** argv) {
```

(continues on next page)

(continued from previous page)

```
  check_help(argc, argv, "cepton_sdk_sample_process_multi capture_path");
  if (!CEPTON_ASSERT(argc >= 2, "Capture path not provided!")) std::exit(1);
  const std::string capture_path = argv[1];

  // Initialize
  auto options = cepton_sdk::create_options();
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());

  // Listen
  FrameAccumulator accumulator;
  CEPTON_CHECK_ERROR(
      callback.listen(&accumulator, &FrameAccumulator::on_image_frame));

  while (!cepton_sdk::capture_replay::is_end()) {
    // Get frame
    if (accumulator.queue.empty())
      CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume_blocking(0.1f));
    const auto frame = accumulator.queue.pop();
    if (!frame) continue;

    // Do processing...
  }

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.6 Process Single

Listing 11: samples/advanced/process_single.cpp

```
/**
 * Sample code for processing offline data from single sensor.
 */
#include <cepton_sdk_api.hpp>

#include "common.hpp"

struct Frame {
  int64_t timestamp;
  cepton_sdk::SensorHandle handle;
  std::vector<cepton_sdk::SensorImagePoint> image_points;
};

int main(int argc, char **argv) {
  check_help(argc, argv, "cepton_sdk_sample_process_single capture_path");
  if (!CEPTON_ASSERT(argc >= 2, "Capture path not provided!")) std::exit(1);
  const std::string capture_path = argv[1];

  // Initialize
  auto options = cepton_sdk::create_options();
  options.frame.mode = CEPTON_SDK_FRAME_COVER;
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
```

(continues on next page)

(continued from previous page)

```
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());

  // Listen
  cepton_sdk::util::SingleConsumerQueue<Frame> queue;
  CEPTON_CHECK_ERROR(
      callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
                          const cepton_sdk::SensorImagePoint *c_image_points) {
        // Add frame to queue
        auto frame = std::make_shared<Frame>();
        frame->timestamp = cepton_sdk::api::get_time();
        frame->handle = handle;
        frame->image_points.insert(frame->image_points.end(), c_image_points,
                                   c_image_points + n_points);
        queue.push(frame);
      }));

  while (!cepton_sdk::capture_replay::is_end()) {
    // Get frame
    if (queue.empty())
      CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume_blocking(0.1f));
    const auto frame = queue.pop();
    if (!frame) continue;

    // Do processing...
  }

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.7 Replay

Listing 12: samples/advanced/replay.cpp

```
/**
 * Sample code for custom packet replaying.
 */
#include <cepton_sdk/capture.hpp>
#include <cepton_sdk_api.hpp>

class CaptureReplay {
 public:
  CaptureReplay(const std::string& path) {
    CEPTON_CHECK_ERROR(m_capture.open_for_read(path));

    CEPTON_CHECK_ERROR(
        cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
                                     CEPTON_SDK_CONTROL_DISABLE_NETWORK));
    CEPTON_CHECK_ERROR(cepton_sdk_clear());
  }

  ~CaptureReplay() {
    m_capture.close();
    if (cepton_sdk_is_initialized()) {
```

(continues on next page)

(continued from previous page)

```
      CEPTON_CHECK_ERROR(cepton_sdk_clear());
    }
  }

  void run() {
    while (true) {
      cepton_sdk::Capture::PacketHeader header;
      const uint8_t* data;
      CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));

      const cepton_sdk::SensorHandle handle =
          (cepton_sdk::SensorHandle)header.ip_v4 |
          CEPTON_SENSOR_HANDLE_FLAG_MOCK;
      CEPTON_CHECK_ERROR(cepton_sdk_mock_network_receive(
          handle, header.timestamp, data, header.data_size));
    }
  }

 private:
  cepton_sdk::Capture m_capture;
};

int main(int argc, char** argv) {
  if (argc < 2) return -1;
  const std::string capture_path = argv[1];

  // Initialize
  auto options = cepton_sdk::create_options();
  options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
  options.frame.mode = CEPTON_SDK_FRAME_COVER;
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));

  // Listen for points
  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  CEPTON_CHECK_ERROR(
      callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
                         const cepton_sdk::SensorImagePoint* c_image_points) {
        std::printf("Received %i points from sensor %lli\n", (int)n_points,
                    (long long)handle);
      }));

  // Run
  CaptureReplay replay(capture_path);
  replay.run();

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

3.3.8 Stray Filter

Listing 13: samples/advanced/stray_filter.cpp

```cpp
/**
 * Sample code for stray filter usage.
 */
#include <vector>

#include <cepton_sdk_api.hpp>

int main(int argc, char **argv) {
  // Initialize
  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize());

  cepton_sdk::api::SensorImageFrameCallback callback;
  CEPTON_CHECK_ERROR(callback.initialize());
  cepton_sdk::util::StrayFilter filter;
  CEPTON_CHECK_ERROR(
      callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
                          const cepton_sdk::SensorImagePoint *c_image_points) {
        // Get sensor
        cepton_sdk::SensorInformation sensor_info;
        CEPTON_CHECK_ERROR(
            cepton_sdk::get_sensor_information(handle, sensor_info));

        // Copy points to buffer
        std::vector<cepton_sdk::SensorImagePoint> image_points;
        image_points.insert(image_points.begin(), c_image_points,
                            c_image_points + n_points);

        // Filter stray.
        filter.init(sensor_info);
        filter.run((int)n_points, image_points.data());
      }));

  // Run
  CEPTON_CHECK_ERROR(cepton_sdk::api::wait(5.0f));

  // Deinitialize
  cepton_sdk::deinitialize().ignore();
}
```

CHAPTER

FOUR

INTERNAL

This page provides a brief description of what goes on inside the SDK.

4.1 Packet Received

Occurs when `cepton_sdk_mock_network_receive` is called.

1. **If the packet is a sensor information packet:**
    - Update the internal stored sensor state. This information can be queried with *cepton_sdk_get_sensor_information()*.
2. **If the packet is a data packet:**
    - If no corresponding sensor information packet has been received, ignore the data packet.
    - Compute points from packet.
    - Apply sensor calibration.
    - Add points to frame accumulator. If frame is complete, emit image frame callback.

4.2 Threads

4.2.1 Networking Thread 0

Created if *CEPTON_SDK_CONTROL_DISABLE_NETWORK* is not set.

- Start listening on the UDP port (default: `8808`).
- Push received packets on a queue for `Thread 1`.

4.2.2 Networking Thread 1

Created if *CEPTON_SDK_CONTROL_DISABLE_NETWORK* is not set.

- Pop packets from queue.
- Call internal version of `cepton_sdk_mock_network_receive`. See `Packet Received`.

4.2.3 Capture Replay Thread 0

Created by *cepton_sdk_capture_replay_resume()*.

- **While running**
  - Read next packet from PCAP file.
  - Sleep to simulate realtime delays.
  - Call `cepton_sdk_mock_network_receive`.

4.3 Concurrency

All SDK getter functions are thread safe, and can be called from callbacks. Other SDK functions are not guaranteed to be thread safe, and can cause deadlock if called from callbacks.

4.4 Minimal SDK

If desired, the following SDK features can be disabled in the SDK and performed manually by the user:

- Capture Replay: *Replay*.
- Frame Accumulation: *Frame Detector*.
- Networking: *Network*.

CHAPTER

FIVE

LEGAL

CHAPTER

SIX

TOOLS

6.1 Cepton Viewer

The Cepton Viewer tool allows viewing raw Cepton sensor data. It is primarily used for sensor debugging.

- View live or replay sensor data.
- View sensor information.
- Update extrinsic sensor transforms.

6.1.1 Downloads

- Windows
- OSX
- Linux

On Windows, a firewall popup will appear the first time Cepton Viewer is launched. Select "Allow access" for both private and public networks.

6.1.2 Main Menu

- Hover over menu items to view tooltips.
- When editing value fields, press `ESC` to undo new value and `ENTER` to accept new value.

6.1.3 Camera Controls

Command	Mouse
Rotate View	`LButton`
Translate View	`RButton`
Zoom View	`MButton`

6.1.4 Workflows

Open Capture

- Drag and drop capture file/folder onto the main window.

OR

- Switch to General tab.
- Select Capture -> Load Capture.

Save Capture

- Switch to General tab.
- Select Capture -> Start Capture.
- Wait for desired duration.
- Select Capture -> Stop Capture.

The capture is saved at ~/Documents/CeptonViewer/<date>/capture_<time>.

Take Screenshot

- Switch to General tab.
- Select Tools -> Screenshot.

The screenshot is saved at ~/Documents/CeptonViewer/<date>/screenshot_<time>.png.

Update Sensor Transforms/Clips

- Switch to Settings tab.
- For each sensor, update the transforms/clips.
- Select File -> Save/Save As....

6.1.5 Keyboard Shortcuts

GUI

Command	Key
Hide GUI	F11

Replay

Command	Key
Pause/Resume	SPACE

View

Command	Key
Reset Camera Translation	`O`
Camera Front View	`1`
Camera Top View	`2`
Camera Side View	`3`

6.2 Cepton Player

The Cepton Player tool allows creating/viewing data captures.

- View live/replay data.
- Capture data.
- Clip/filter data.
- Measure LiDAR points.
- Export LiDAR points.

6.2.1 Install

Requirements:

- Python 3

Install `cepton_alg`

```
pip3 install --user -U cepton_alg
```

To launch, run

```
cepton_player.py
```

6.2.2 Workflows

Open Capture

- Drag and drop capture file/folder onto the main window.

`OR`

- Select `File` -> `Open Replay`.

6.2.3 Camera Controls

Command	Mouse
Rotate View	`LButton`
Translate View	`RButton`
Zoom View	`Scroll`

6.2.4 Cursors

Interactive cursors use a combination of `CTRL` + `LButton`/`RButton`.

Distance Ruler

Command	Key
Enable/Disable	`D`
Measure	`CTRL` + `LButton`

Angle Ruler

Command	Key
Enable/Disable	`A`
Measure	`CTRL` + `LButton`
Set Center	`CTRL` + `RButton`

Point Selection

Command	Key
Enable/Disable	`S`
Select	`CTRL` + `LButton`

6.2.5 Keyboard Shortcuts

Replay

Command	Key
Pause/Resume	`SPACE`
Next Frame	`N`

View

Command	Key
Reset View Origin	`0`
Front View	`1`
Top View	`2`
Side View	`3`

6.3 Cepton Export

The Cepton Export tool allows exporting LiDAR points in various file formats.

6.3.1 Install

Requirements:

- Python 3

Install the Cepton Python SDK

```
pip3 install --user -U cepton_sdk[export]
```

For usage, run

```
cepton_export.py -h
```

CHAPTER

SEVEN

API REFERENCE

7.1 Errors

7.1.1 Types

class SensorError : **public** runtime_error
Error returned by most functions.

Implicitly convertible from/to `SensorErrorCode`. Getter functions do not return an error, because they cannot fail. Will call `CEPTON_ASSERT` if nonzero error is not used (call `ignore` to manually use error).

Public Functions

SensorError(SensorErrorCode *code*, **const** std::string &*msg*)
Create *SensorError* class object with SensorErrorCode and error message.

SensorError(SensorErrorCode *code*)
Create *SensorError* class object with error code.

SensorError()
SensorError class default constructor.

~SensorError()
SensorError class destructor.

SensorError(**const** *SensorError* &*other*)
Create *SensorError* object using *SensorError* object.

SensorError &**operator=**(**const** *SensorError* &*other*)
SensorError class assignment operator.

bool **used**() **const**
Internal use only.

const *SensorError* &**ignore**() **const**
Mark error as used.

const char ***what**() **const**

const std::string &**msg**() **const**
Returns error message.

SensorErrorCode **code**() **const**
Returns error code.

operator SensorErrorCode() **const**
Implicitly convert to `SensorErrorCode`.

operator bool() **const**
Returns `false` if error code is `CEPTON_SUCCESS`, true otherwise.

const std::string **name**() **const**
Returns error code name.

bool **is_error**() **const**
Returns true if parent object is an error.

bool **is_fault**() **const**
Returns true if parent object is a fault.

typedef int32_t **CeptonSensorErrorCode**
Error code returned by most library functions.

Must call `cepton_sdk_get_error` if nonzero error code is returned.

enum _CeptonSensorErrorCode
Values:

CEPTON_SUCCESS = 0
No error.

CEPTON_ERROR_GENERIC = -1
Generic error.

CEPTON_ERROR_OUT_OF_MEMORY = -2
Failed to allocate heap memory.

CEPTON_ERROR_SENSOR_NOT_FOUND = -4
Could not find sensor.

CEPTON_ERROR_SDK_VERSION_MISMATCH = -5
SDK version mismatch.

CEPTON_ERROR_COMMUNICATION = -6
Networking error.

CEPTON_ERROR_TOO_MANY_CALLBACKS = -7
Callback already set.

CEPTON_ERROR_INVALID_ARGUMENTS = -8
Invalid value or uninitialized struct.

CEPTON_ERROR_ALREADY_INITIALIZED = -9
Already initialized.

CEPTON_ERROR_NOT_INITIALIZED = -10
Not initialized.

CEPTON_ERROR_INVALID_FILE_TYPE = -11
Invalid file type.

CEPTON_ERROR_FILE_IO = -12
File IO error.

CEPTON_ERROR_CORRUPT_FILE = -13
Corrupt/invalid file.

CEPTON_ERROR_NOT_OPEN = -14
Not open.

CEPTON_ERROR_EOF = -15
End of file.

CEPTON_FAULT_INTERNAL = -1000
Internal sensor parameter out of range.

CEPTON_FAULT_EXTREME_TEMPERATURE = -1001
Extreme sensor temperature fault.

CEPTON_FAULT_EXTREME_HUMIDITY = -1002
Extreme sensor humidity fault.

CEPTON_FAULT_EXTREME_ACCELERATION = -1003
Extreme sensor acceleration fault.

CEPTON_FAULT_ABNORMAL_FOV = -1004
Abnormal sensor FOV fault.

CEPTON_FAULT_ABNORMAL_FRAME_RATE = -1005
Abnormal sensor frame rate fault.

CEPTON_FAULT_MOTOR_MALFUNCTION = -1006
Sensor motor malfunction fault.

CEPTON_FAULT_LASER_MALFUNCTION = -1007
Sensor laser malfunction fault.

CEPTON_FAULT_DETECTOR_MALFUNCTION = -1008
Sensor detector malfunction fault.

7.1.2 Methods

const char ***cepton_get_error_code_name**(*CeptonSensorErrorCode error_code*)
Returns error code name string.

Returns empty string if error code is invalid.

Return Error code name string. Owned by SDK. Valid until next SDK call in current thread.

int **cepton_is_error_code**(*CeptonSensorErrorCode error_code*)
Returns whether `error_code` is of the form `CEPTON_ERROR_*`.

int **cepton_is_fault_code**(*CeptonSensorErrorCode error_code*)
Returns whether `error_code` is of the form `CEPTON_FAULT_*`.

CeptonSensorErrorCode **cepton_sdk_get_error**(**const** char ***error_msg*)
Returns and clears last sdk error.

Return Error code.

Parameters

- `error_msg`: Returned error message string. Owned by the SDK, and valid until next SDK call in the current thread.

7.2 Setup

7.2.1 Types

typedef uint32_t **CeptonSDKControl**
SDK setup flags.

enum _CeptonSDKControl
Values:

CEPTON_SDK_CONTROL_DISABLE_NETWORK = 1 << 1
Disable networking operations.

Useful for running multiple instances of sdk in different processes. Must pass packets manually to `cepton_sdk::mock_network_receive`.

CEPTON_SDK_CONTROL_DISABLE_IMAGE_CLIP = 1 << 2
Disable marking image clipped points as invalid.

Internal use. Does not affect number of points returned.

CEPTON_SDK_CONTROL_DISABLE_DISTANCE_CLIP = 1 << 3
Disable marking distance clipped points as invalid.

Internal use. Does not affect number of points returned.

CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS = 1 << 4
Enable multiple returns.

When set, *cepton_sdk::SensorInformation::return_count* will indicate the number of returns per laser. Can only be set at SDK initialization.

CEPTON_SDK_CONTROL_ENABLE_STRAY_FILTER = 1 << 5
Enable marking stray points as invalid (measurement noise).

Uses `cepton_sdk::util::StrayFilter` to mark points invalid.

Does not affect number of points returned.

CEPTON_SDK_CONTROL_HOST_TIMESTAMPS = 1 << 6
Always use packet timestamps (disable GPS/PTP timestamps).

CEPTON_SDK_CONTROL_ENABLE_CROSSTALK_FILTER = 1 << 7
Enable marking crosstalk points as invalid.

Not implemented.

typedef uint32_t **CeptonSDKFrameMode**
Controls frequency of points being reported.

enum _CeptonSDKFrameMode
Values:

CEPTON_SDK_FRAME_STREAMING = 0
Report points by packet.

CEPTON_SDK_FRAME_TIMED = 1
Report points at fixed time intervals.

Interval controlled by *CeptonSDKFrameOptions::length*.

CEPTON_SDK_FRAME_COVER = 2
Report points when the field of view is covered once.

Use this for a fast frame rate.

- For Sora series, detects scanline (left-to-right or right-to-left).
- For HR80 series, detects half scan cycle (left-to-right or right-to-left).
- For Vista series, detects half scan cycle.

CEPTON_SDK_FRAME_CYCLE = 3
Report points when the scan pattern goes through a full cycle.

Use this for a consistent, repeating frame. Typically 2x longer frame than `CEPTON_SDK_FRAME_COVER` mode.

CEPTON_SDK_FRAME_MODE_MAX = 3

struct CeptonSDKFrameOptions
SDK frame options.

Must use `cepton_sdk_create_frame_options` to create.

Public Members

size_t **signature**
Internal use only.

CeptonSDKFrameMode **mode**
Default: `CEPTON_SDK_FRAME_STREAMING`.

float **length**
Frame length [seconds].

Default: 0.05. Only used if `mode=CEPTON_SDK_FRAME_TIMED`.

struct *CeptonSDKFrameOptions* **cepton_sdk_create_frame_options**()
Create frame options.

struct CeptonSDKOptions
SDK initialization options.

Must call `cepton_sdk_create_options` to create.

Public Members

size_t **signature**
Internal use only.

CeptonSDKControl **control_flags**
Default: 0.

struct *CeptonSDKFrameOptions* **frame**

uint16_t **port**
Network listen port. Default: 8808.

struct *CeptonSDKOptions* **cepton_sdk_create_options**()
Create SDK options.

typedef void (*`cepton_sdk::`**FpSensorErrorCallback**)(SensorHandle handle, SensorErrorCode error_code, **const** char *error_msg, **const** void *error_data, size_t error_data_size, void *user_data)

7.2.2 Methods

const char ***cepton_sdk_get_version_string**()
Returns library version string.

This is different from `CEPTON_SDK_VERSION`.

Return Version string. Owned by SDK. Valid until next SDK call in current thread.

int **cepton_sdk_get_version_major**()
Returns library version major.

int **cepton_sdk_get_version_minor**()
Returns library version minor.

CeptonSensorErrorCode **cepton_sdk_initialize**(int *ver*, **const struct** *CeptonSDKOptions* ***const** *options*, FpCeptonSensorErrorCallback *cb*, void ***const** *user_data*)
Initializes settings and networking.

Must be called before any other sdk function listed below.

Parameters

- `ver`: `CEPTON_SDK_VERSION`
- `options`: SDK options.
- `cb`: Error callback.
- `user_data`: Error callback user instance pointer.

CeptonSensorErrorCode **cepton_sdk_deinitialize**()
Resets everything and deallocates memory.

CeptonSensorErrorCode **cepton_sdk_clear**()
Clears sensors.

Use when loading/unloading capture file.

CeptonSensorErrorCode **cepton_sdk_set_control_flags**(*CeptonSDKControl* *mask*, *CeptonSDKControl* *flags*)
Sets SDK control flags.

Parameters

- `mask`: Bit mask for selecting flags to change.
- `flags`: Bit flag values.

CeptonSDKControl **cepton_sdk_get_control_flags**()
Returns SDK control flag.

int **cepton_sdk_has_control_flag**(*CeptonSDKControl* *flag*)
Returns whether SDK control flag is set.

uint16_t **cepton_sdk_get_port**()
Returns network listen port.

CeptonSensorErrorCode **cepton_sdk_set_port**(uint16_t *port*)
Sets network listen port.

Default: 8808.

CeptonSensorErrorCode **cepton_sdk_set_frame_options**(**const struct** *CeptonSDKFrameOptions* ***const** *options*)
Sets frame options.

CeptonSDKFrameMode **cepton_sdk_get_frame_mode**()
Returns frame mode.

float **cepton_sdk_get_frame_length**()
Returns frame length.

7.3 Sensors

7.3.1 Types

typedef uint64_t **CeptonSensorHandle**
Sensor identifier.

Generated from sensor IP address.

typedef uint16_t **CeptonSensorModel**
Sensor model.

enum _CeptonSensorModel
Values:

HR80W = 3

HR80T_R2 = 6

VISTA_860_GEN2 = 7

VISTA_X120 = 10

SORA_P60 = 11

VISTA_P60 = 12

VISTA_X15 = 13

VISTA_P90 = 14

SORA_P90 = 15

VISTA_P61 = 16

SORA_P61 = 17

VISTA_H120 = 18

CEPTON_SENSOR_MODEL_MAX = 18

struct CeptonSensorInformation
Sensor information struct.

Returned by `cepton_sdk_get_sensor_information*`.

Public Members

CeptonSensorHandle **handle**
Sensor identifier (generated from IP address).

uint64_t **serial_number**
Sensor serial number.

char **model_name**[28]
Full sensor model name.

CeptonSensorModel **model**
Sensor model.

uint16_t **reserved**

char **firmware_version**[28]
Firmware version string.

uint8_t **major**
Major firmware version.

uint8_t **minor**
Minor firmware version.

uint8_t **unused**[2]

struct *CeptonSensorInformation*::**[anonymous] formal_firmware_version**
Firmware version struct.

float **last_reported_temperature**
[celsius].

float **last_reported_humidity**
[%].

float **last_reported_age**
[hours].

float **measurement_period**
Time between measurements [seconds].

int64_t **ptp_ts**
PTP time [microseconds].

uint8_t **gps_ts_year**
(0-99) (e.g. 2017 -> 17)

uint8_t **gps_ts_month**
(1-12)

uint8_t **gps_ts_day**
(1-31)

uint8_t **gps_ts_hour**
(0-23)

uint8_t **gps_ts_min**
(0-59)

uint8_t **gps_ts_sec**
(0-59)

uint8_t **return_count**
Number of returns per measurement.

uint8_t **segment_count**
Number of image segments.

uint32_t **flags**
Bit flags.

uint32_t **is_mocked**
Created by capture replay.

uint32_t **is_pps_connected**
GPS PPS is available.

uint32_t **is_nmea_connected**
GPS NMEA is available.

uint32_t **is_ptp_connected**
PTP is available.

uint32_t **is_calibrated**
Calibration loaded.

uint32_t **is_over_heated**
Hit temperature limit.

uint32_t **is_sync_firing_enabled**
Sync fire enabled (disabled by default).

union *CeptonSensorInformation*::**[anonymous] [anonymous]**

7.3.2 Methods

int **cepton_is_sora** (*CeptonSensorModel model*)
Returns whether sensor model is of the form `SORA_*`.

int **cepton_is_hr80** (*CeptonSensorModel model*)
Returns whether sensor model is of the form `HR80_*`.

int **cepton_is_vista** (*CeptonSensorModel model*)
Returns whether sensor model is of the form `VISTA_*`.

size_t **cepton_sdk_get_n_sensors** ()
Get number of sensors attached. Use to check for new sensors. Sensors are not deleted until deinitialization.

CeptonSensorErrorCode **cepton_sdk_get_sensor_handle_by_serial_number** (uint64_t *serial_number*, *CeptonSensorHandle* ***const** *handle*)
Looks up sensor handle by serial number.

Returns error if sensor not found.

Parameters

- `serial_number`: Sensor serial number.
- `handle`: Sensor handle.

CeptonSensorErrorCode **cepton_sdk_get_sensor_information_by_index** (size_t *idx*, **struct** *CeptonSensorInformation* ***const** *info*)
Returns sensor information by sensor index.

Useful for getting information for all sensors. Valid indices are in range [0, *cepton_sdk_get_n_sensors()*).

Returns error if index invalid.

Parameters

- `idx`: Sensor index. Returns error if invalid.
- `info`: Sensor information.

CeptonSensorErrorCode **cepton_sdk_get_sensor_information**(*CeptonSensorHandle* *handle*, **struct** *CeptonSensorInformation* ***const** *info*)

Returns sensor information by sensor handle.

Parameters

- `handle`: Sensor handle. Returns error if invalid.
- `info`: Sensor information.

7.4 Points

7.4.1 Types

struct CeptonSensorImagePoint

Point in pinhole image coordinates (focal length = 1).

To convert to 3d point, use *cepton_sdk::util::convert_sensor_image_point_to_point*.

Public Members

int64_t **timestamp**
Unix time [microseconds].

float **image_x**
x image coordinate.

float **distance**
Distance [meters].

float **image_z**
z image coordinate.

float **intensity**
Diffuse reflectance (normal: [0-1], retroreflective: >1).

CeptonSensorReturnType **return_type**
Return type flags.

uint8_t **flags**
Bit flags.

uint8_t **valid**
If `false`, then `distance` and `intensity` are invalid.

uint8_t **saturated**
If `true`, then `intensity` is invalid, and the `distance` is innacurate.

union *CeptonSensorImagePoint*::**[anonymous] [anonymous]**

uint8_t **segment_id**

uint8_t **reserved**[1]

7.4.2 Methods

typedef void (***FpCeptonSensorImageDataCallback**) (*CeptonSensorHandle* handle, size_t n_points, **const struct** *CeptonSensorImagePoint* *c_points, void *user_data)

Callback for receiving image points.

Set the frame options to control the callback rate.

Parameters

- `handle`: Sensor handle.
- `n_points`: Points array size.
- `c_points`: Points array. Owned by SDK.
- `user_data`: User instance pointer.

CeptonSensorErrorCode **cepton_sdk_listen_image_frames** (*FpCeptonSensorImageDataCallback* *cb*, void ***const** *user_data*)

Sets image frame callback.

Returns points at frequency specified by *`cepton_sdk::FrameOptions::mode`*. Each frame contains all possible points (use *`cepton_sdk::SensorImagePoint::valid`* to filter points). Points are ordered by measurement, segment, and return:

```
measurement_count = n_points / (segment_count * return_count)
idx = ((i_measurement) * segment_count + i_segment) * return_count + i_return
```

Returns error if callback already registered.

CeptonSensorErrorCode **cepton_sdk_unlisten_image_frames** ()
Clears image frame callback.

7.5 Capture Replay

PCAP capture file replay. Functions are not thread safe, and should only be called from the main thread.

int **cepton_sdk_capture_replay_is_open** ()
Returns whether capture replay is open.

CeptonSensorErrorCode **cepton_sdk_capture_replay_open** (**const** char ***const** *path*)
Opens capture replay.

Must be called before any other replay functions listed below.

Parameters

- `path`: Path to PCAP capture file.

CeptonSensorErrorCode **cepton_sdk_capture_replay_close**()
Closes capture replay.

const char ***cepton_sdk_capture_replay_get_filename**()
Returns capture replay file name.

int64_t **cepton_sdk_capture_replay_get_start_time**()
Returns capture start Unix timestamp [microseconds].

float **cepton_sdk_capture_replay_get_position**()
Returns capture file position [seconds].

float **cepton_sdk_capture_replay_get_length**()
Returns capture file length [seconds].

int **cepton_sdk_capture_replay_is_end**()
Returns whether at end of capture file.

This is only relevant when using `resume_blocking` methods.

CeptonSensorErrorCode **cepton_sdk_capture_replay_seek**(float *position*)
Seek to capture file position [seconds].

Parameters

- `position`: Seek position in range [0.0, capture length).

CeptonSensorErrorCode **cepton_sdk_capture_replay_set_enable_loop**(int *enable_loop*)
Sets capture replay looping.

If enabled, replay will automatically rewind at end.

int **cepton_sdk_capture_replay_get_enable_loop**()
Returns whether capture replay looping is enabled.

CeptonSensorErrorCode **cepton_sdk_capture_replay_set_speed**(float *speed*)
Sets speed multiplier for asynchronous replay.

float **cepton_sdk_capture_replay_get_speed**()
Returns capture replay speed.

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume_blocking_once**()
Replay next packet in current thread without sleeping.

Pauses replay thread if running.

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume_blocking**(float *duration*)
Replay multiple packets synchronously.

No sleep between packets. Pauses replay thread if running.

Parameters

- `duration`: Duration to replay. Must be non-negative.

int **cepton_sdk_capture_replay_is_running**()
Returns true if replay thread is running.

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume**()
Resumes asynchronous replay thread.

Packets are replayed in realtime. Replay thread sleeps in between packets.

CeptonSensorErrorCode **cepton_sdk_capture_replay_pause**()
Pauses asynchronous replay thread.

7.6 Networking

Network callback for debugging.

7.6.1 Types

typedef void (***FpCeptonNetworkReceiveCallback**)(*CeptonSensorHandle* handle, int64_t timestamp, **const** uint8_t *buffer, size_t buffer_size, void *user_data)

Callback for receiving network packets.

Returns error if callback already set.

Parameters

- `handle`: Sensor handle.
- `timestamp`: Packet Unix timestamp [microseconds].
- `buffer`: Packet bytes.
- `buffer_size`: Buffer size.
- `user_data`: User instance pointer.

7.6.2 Methods

CeptonSensorErrorCode **cepton_sdk_listen_network_packet**(*FpCeptonNetworkReceiveCallback* *cb*, void ***const** *user_data*)

Sets network packets callback.

For internal use.

Returns error if callback already registered.

Parameters

- `cb`: Callback.
- `user_data`: User instance pointer.

CeptonSensorErrorCode **cepton_sdk_unlisten_network_packet**()
Clears network packet callback.

7.7 Serial

Serial callback. Primarily used for receiving data from GPS/INS attached to sensor.

7.7.1 Types

typedef void (***FpCeptonSerialReceiveCallback**)(*CeptonSensorHandle* handle, **const** char *str, void *user_data)

Callback for receiving serial data (e.g. NMEA).

Parameters

- `handle`: Sensor handle.
- `str`: Serial line string. Owned by SDK.
- `user_data`: User instance pointer.

7.7.2 Methods

CeptonSensorErrorCode **cepton_sdk_listen_serial_lines**(*FpCeptonSerialReceiveCallback* *cb*, void ***const** *user_data*)

Sets serial line callback.

Useful for listening to NMEA data from GPS attached to sensor. Each callback contains 1 line of serial data (including newline characters).

Returns error if callback already registered.

Parameters

- `cb`: Callback.
- `user_data`: User instance pointer.

CeptonSensorErrorCode **cepton_sdk_unlisten_serial_lines**()

Clears serial line callback.

7.8 C++

High level C++ API for prototyping (`cepton_sdk.hpp`, `cepton_sdk_api.hpp`). Methods are agnostic to live/replay mode.

bool `cepton_sdk::api::`**is_live**()

Returns whether capture replay is not open.

bool `cepton_sdk::api::`**is_end**()

Returns whether capture replay is at the end and enable loop is false.

int64_t `cepton_sdk::api::`**get_time**()

Returns live or capture replay time.

SensorError `cepton_sdk::api::`**wait**(float *t_length* = -1.0f)

Sleeps or resumes capture replay for duration.

If `t_length < 0`, then waits forever.

7.8.1 Errors

CEPTON_PROCESS_ERROR(code)
Add context to error.

CEPTON_CHECK_ERROR(code)
If error, raise.

CEPTON_LOG_ERROR(code)
If error, print.

CEPTON_RETURN_ERROR(code)
If error, return.

7.8.2 Setup

SensorError `cepton_sdk::api::`**initialize**(Options *options* = create_options(), **const** std::string &*capture_path* = "", bool *enable_wait* = false)
Initialize SDK and optionally starts capture replay.

If `enable_wait` is true, waits a few seconds to initialize sensors.

SensorError `cepton_sdk::api::`**open_replay**(**const** std::string &*capture_path*, bool *enable_wait* = false)
Opens capture replay.

If `enable_wait` is true, replays a few seconds to intialize sensors.

bool `cepton_sdk::api::`**has_control_flags**(Control *mask*)
Returns whether indicated control flags are set.

SensorError `cepton_sdk::api::`**enable_control_flags**(Control *mask*, bool *tf*)
Enables/disables indicated control flags.

class SensorErrorCallback : **public** cepton_sdk::util::Callback<SensorHandle, **const** *SensorError*&>
Callback for sensor errors.

Public Static Functions

static void **global_on_callback**(SensorHandle *handle*, SensorErrorCode *error_code*, **const** char **error_msg*, **const** void ***const** *error_data*, size_t *error_data_size*, void ***const** *instance*)

class SensorImageFrameCallback : **public** cepton_sdk::util::Callback<SensorHandle, std::size_t, **const** SensorImagePoi
Callback for image frames.

Must call `initialize` before use.

Public Functions

~SensorImageFrameCallback()
SensorImageFrameCallback class destructor.

SensorError **initialize**()
Initializes *SensorImageFrameCallback* object.

SensorError **deinitialize**()
Deinitializes *SensorImageFrameCallback* object.

bool **is_initialized**() **const**
Returns true if *SensorImageFrameCallback* is initialized.

7.8.3 Sensors

bool cepton_sdk::api::**has_sensor_by_serial_number**(uint64_t *serial_number*)
Returns whether SDK has sensor with serial number.

SensorError cepton_sdk::api::**get_sensor_information_by_serial_number**(uint64_t *serial_number*, SensorInformation &*info*)
Returns sensor information by serial number.

Returns error if sensor not found.

std::vector<uint64_t> cepton_sdk::api::**get_sensor_serial_numbers**()
Returns serial numbers for all sensors.

7.9 Utilities

Utility functions and classes for prototyping (*cepton_sdk_util.hpp*).

7.9.1 Common

int64_t cepton_sdk::util::**get_timestamp_usec**()
Returns current unix timestamp [microseconds].

This is the timestamp format used by all sdk functions.

7.9.2 Points

struct SensorPoint
3d point class.

Can't subclass from SensorImagePoint, needs to be POD.

Public Members

int64_t **timestamp**
Unix time [microseconds].

float **image_x**
x image coordinate.

float **distance**
Distance [meters].

float **image_z**
z image coordinate.

float **intensity**
Diffuse reflectance.

CeptonSensorReturnType **return_type**
Strongest or farthest return.

uint8_t **flags**
Bit flags.

uint8_t **valid**
If `false`, then the distance and intensity are invalid.

uint8_t **saturated**
If `true`, then the intensity is invalid. Also, the distance is valid, but inaccurate.

union cepton_sdk::util::*SensorPoint*::**[anonymous] [anonymous]**

float **x**
x cartesian coordinate

float **y**
y cartesian coordinate

float **z**
z cartesian coordinate

void `cepton_sdk::util::`**convert_sensor_image_point_to_point**(**const** SensorImagePoint &*image_point*, *SensorPoint* &*point*)
Convenience method to convert `cepton_sdk::SensorImagePoint` to `cepton_sdk::SensorPoint`.

7.9.3 Callbacks

template<typename ...**TArgs**>
class Callback
Expands SDK callback functionality.

Allows for multiple callbacks to be registered. Allows for registering lambdas and member functions. See `samples/basic.cpp`.

Public Functions

void **clear**()
Clear all listeners.

SensorError **listen**(**const** std::function<void(TArgs...)> &*func*, uint64_t ***const** *id* = nullptr)
Register a callback function.

Parameters

- `func`: *Callback* function.
- `id`: Identifier used for `unlisten`.

template<typename **TClass**>
SensorError **listen**(*TClass* ***const** *instance*, MemberFunction<*TClass*, void(TArgs...)> *func*, uint64_t ***const** *id* = nullptr)
Register instance member function.

Parameters

- `instance`: Parent class instance pointer.
- `func`: *Callback* function pointer.
- `id`: Identifier used for `unlisten`.

SensorError **unlisten**(uint64_t *id*)
Unregister function.

Parameters

- `id`: Identifier returned by `listen`.

void **operator()**(TArgs... *args*) **const**
Emit callback.

Calls all registered functions with `args`.

Public Static Functions

static void **global_on_callback**(TArgs... *args*, void ***const** *instance*)
Used for registering as c callback.

7.9.4 Frames

template<typename **TData** = bool>
class FrameDetector : **public** cepton_sdk::util::internal::FrameDetectorBase<*TData*>
Detects frames in streaming sensor data.

`Result::type`

- Sora: 0=left-right, 1=right-left
- HR80: 0=left-right, 1=right-left
- Vista: undefined

Public Functions

FrameDetector(**const** SensorInformation &*sensor_info*)
FrameDetector class constructor passing in SensorInformation.

const FrameOptions &**get_options**() **const**
Returns frame options for *FrameDetector*.

SensorError **set_options**(**const** FrameOptions &*options*)
Set frame options.

void **reset**()
Completely resets detector.

Only use if also clearing points accumulator.

bool **update**(**const** SensorImagePoint &*point*, **const** TData &*data* = TData())
Returns true if frame found.

Automatically resets after frame is found.

class FrameAccumulator
Accumulates image points, and emits frames to callback.

See `samples/frame.cpp`.

Public Functions

FrameAccumulator(**const** SensorInformation &*sensor_info*)
FrameAccumulator class constructor passing in SensorInformation.

FrameOptions **get_options**() **const**
Return frame options for *FrameAccumulator*.

SensorError **set_options**(**const** FrameOptions &*options*)
Set options for *FrameAccumulator*.

void **clear**()

void **add_points**(std::size_t *n_points*, **const** SensorImagePoint ***const** *image_points*)

Public Members

Callback<std::size_t, **const** SensorImagePoint *> **callback**
Frames callback.

7.9.5 Organizer

struct OrganizedCloud
The *OrganizedCloud* struct An organized version of the cepton point cloud.

Public Functions

int **getIndex**(int *row*, int *col*, int *n_return*)
getIndex Returns the index of the point corresponding to the inputed row, col and return number.

Return

Parameters

- `[in] row`: Row index
- `[in] col`: Col index
- `[in] n_return`: Return index

Public Members

int64_t **timestamp_start**
timestamp_start The time of the oldest point in the cloud

int64_t **timestamp_end**
timestamp_end The time of the newest point in the cloud

int **height**
height Height of the cloud. Represents how many rows there are in the cloud

int **width**
width Width of the cloud. Represents how many columns there are in the cloud

int **n_returns**
n_returns Number of return represented by the cloud.

std::vector<*CellInfo*> **info_cells**
info_cells Vector of cell info which provide information about the matching points

std::vector<*CeptonSensorImagePoint*> **points**
points Vector of organized points. Stored in Return, Row, Col order. So to get a point at row 10, col 15, return 1 would be points[(row * width

- col) n_returns + return

struct CellInfo
The *CellInfo* struct.

Public Members

bool **occupied_cell** = false
occupied_cell Is the cell at this index occupied with a point. If false can't assume this represents free space.

int **original_index** = -1
original_index Index of the point that was used to generate the organized point. Can be used to match back with orginial data if required. Should only be use if occupied_cell is true.

class Organizer
The *Organizer* class Performs organization on cepton unorganized points. Creates an angular grid, places each point within that grid and outputs a point for each location in the grid in a row/col format. Thread safe. Defaults to a 0.4deg spaced grid.

Public Types

enum OrganizerMode
Values:

RECENT
Output the most recent point from the frame that fell within the grid

CENTER
Output the center of the grid. Uses median point distance.

Public Functions

Organizer(cepton_sdk::SensorInformation *sensor_info*)
Organizer.

Parameters

- `sensor_info`: Sensor info for organizer. Used to set min/max angles

void **organize_points**(**const** int *num_points_in*, **const** int *n_returns*, **const** *CeptonSensorImagePoint* ***const** *unorganized_points*, cepton_sdk::util::*OrganizedCloud* &*organized_points*)
organize_points

Parameters

- `[in] num_points_in`: Number of unorganized points
- `[in] n_returns`: Number of returns
- `[in] unorganized_points`: Unorganized points to proces
- `[out] organized_points`: Points in organized form

void **mode**(*OrganizerMode mode*)
mode

Parameters

- `mode`: Change the mode of the organizer. [RECENT] Points are the most recent which fill within the grid. [CENTER] Points outputted are at the center of the grid. More even spacing but less accurate.

void **binSize**(float *bin_size*)
binSize Change the bin size of the organizer

Parameters

- `bin_size`: The horizontal and vertical bin size to set. In radians

void **settings**(*OrganizerSettings organizer_settings*)
settings

Parameters

- `organizer_settings`: Change organizer settings

Organizer::*OrganizerSettings* **settings**()
settings

Return The settings the organizer is using

struct OrganizerSettings

Public Members

float **horizontal_range_radians** = to_radians(70.f)

float **vertical_range_radians** = to_radians(30.f)

float **horizontal_bin_size_radians** = to_radians(0.4f)

float **vertical_bin_size_radians** = to_radians(0.4f)

OrganizerMode **mode** = *OrganizerMode*::RECENT

INDEX

Symbols

_CeptonSDKControl (*C++ enum*), 36
_CeptonSDKFrameMode (*C++ enum*), 36
_CeptonSensorErrorCode (*C++ enum*), 34
_CeptonSensorModel (*C++ enum*), 39

C

CEPTON_CHECK_ERROR (*C macro*), 47
CEPTON_ERROR_ALREADY_INITIALIZED (*C++ enumerator*), 34
CEPTON_ERROR_COMMUNICATION (*C++ enumerator*), 34
CEPTON_ERROR_CORRUPT_FILE (*C++ enumerator*), 34
CEPTON_ERROR_EOF (*C++ enumerator*), 35
CEPTON_ERROR_FILE_IO (*C++ enumerator*), 34
CEPTON_ERROR_GENERIC (*C++ enumerator*), 34
CEPTON_ERROR_INVALID_ARGUMENTS (*C++ enumerator*), 34
CEPTON_ERROR_INVALID_FILE_TYPE (*C++ enumerator*), 34
CEPTON_ERROR_NOT_INITIALIZED (*C++ enumerator*), 34
CEPTON_ERROR_NOT_OPEN (*C++ enumerator*), 35
CEPTON_ERROR_OUT_OF_MEMORY (*C++ enumerator*), 34
CEPTON_ERROR_SDK_VERSION_MISMATCH (*C++ enumerator*), 34
CEPTON_ERROR_SENSOR_NOT_FOUND (*C++ enumerator*), 34
CEPTON_ERROR_TOO_MANY_CALLBACKS (*C++ enumerator*), 34
CEPTON_FAULT_ABNORMAL_FOV (*C++ enumerator*), 35
CEPTON_FAULT_ABNORMAL_FRAME_RATE (*C++ enumerator*), 35
CEPTON_FAULT_DETECTOR_MALFUNCTION (*C++ enumerator*), 35
CEPTON_FAULT_EXTREME_ACCELERATION (*C++ enumerator*), 35
CEPTON_FAULT_EXTREME_HUMIDITY (*C++ enumerator*), 35
CEPTON_FAULT_EXTREME_TEMPERATURE (*C++ enumerator*), 35
CEPTON_FAULT_INTERNAL (*C++ enumerator*), 35
CEPTON_FAULT_LASER_MALFUNCTION (*C++ enumerator*), 35
CEPTON_FAULT_MOTOR_MALFUNCTION (*C++ enumerator*), 35
cepton_get_error_code_name (*C++ function*), 35
cepton_is_error_code (*C++ function*), 35
cepton_is_fault_code (*C++ function*), 35
cepton_is_hr80 (*C++ function*), 41
cepton_is_sora (*C++ function*), 41
cepton_is_vista (*C++ function*), 41
CEPTON_LOG_ERROR (*C macro*), 47
CEPTON_PROCESS_ERROR (*C macro*), 47
CEPTON_RETURN_ERROR (*C macro*), 47
cepton_sdk::api::enable_control_flags (*C++ function*), 47
cepton_sdk::api::get_sensor_information_by_serial_n (*C++ function*), 48
cepton_sdk::api::get_sensor_serial_numbers (*C++ function*), 48
cepton_sdk::api::get_time (*C++ function*), 46
cepton_sdk::api::has_control_flags (*C++ function*), 47
cepton_sdk::api::has_sensor_by_serial_number (*C++ function*), 48
cepton_sdk::api::initialize (*C++ function*), 47
cepton_sdk::api::is_end (*C++ function*), 46
cepton_sdk::api::is_live (*C++ function*), 46
cepton_sdk::api::open_replay (*C++ function*), 47
cepton_sdk::api::SensorErrorCallback (*C++ class*), 47
cepton_sdk::api::SensorErrorCallback::global_on_cal (*C++ function*), 47
cepton_sdk::api::SensorImageFrameCallback (*C++ class*), 47
cepton_sdk::api::SensorImageFrameCallback::~Sensorl (*C++ function*), 47

cepton_sdk::api::SensorImageFrameCallback::deinitialize (C++ function), 47
cepton_sdk::api::SensorImageFrameCallback::initialize (C++ function), 47
cepton_sdk::api::SensorImageFrameCallback::is_initialized (C++ function), 48
cepton_sdk::api::wait (C++ function), 46
cepton_sdk::FpSensorErrorCallback (C++ type), 37
cepton_sdk::SensorError (C++ class), 33
cepton_sdk::SensorError::~SensorError (C++ function), 33
cepton_sdk::SensorError::code (C++ function), 33
cepton_sdk::SensorError::ignore (C++ function), 33
cepton_sdk::SensorError::is_error (C++ function), 34
cepton_sdk::SensorError::is_fault (C++ function), 34
cepton_sdk::SensorError::msg (C++ function), 33
cepton_sdk::SensorError::name (C++ function), 34
cepton_sdk::SensorError::operator bool (C++ function), 34
cepton_sdk::SensorError::operator SensorErrorCode (C++ function), 34
cepton_sdk::SensorError::operator= (C++ function), 33
cepton_sdk::SensorError::SensorError (C++ function), 33
cepton_sdk::SensorError::used (C++ function), 33
cepton_sdk::SensorError::what (C++ function), 33
cepton_sdk::util::Callback (C++ class), 49
cepton_sdk::util::Callback::clear (C++ function), 49
cepton_sdk::util::Callback::global_on_callback (C++ function), 50
cepton_sdk::util::Callback::listen (C++ function), 49
cepton_sdk::util::Callback::operator() (C++ function), 50
cepton_sdk::util::Callback::unlisten (C++ function), 50
cepton_sdk::util::convert_sensor_image_points_to_xyz (C++ function), 49
cepton_sdk::util::FrameAccumulator (C++ class), 50
cepton_sdk::util::FrameAccumulator::add_points (C++ function), 51
cepton_sdk::util::FrameAccumulator::callback (C++ member), 51
cepton_sdk::util::FrameAccumulator::clear (C++ function), 51
cepton_sdk::util::FrameAccumulator::FrameAccumulator (C++ function), 51
cepton_sdk::util::FrameAccumulator::get_options (C++ function), 51
cepton_sdk::util::FrameAccumulator::set_options (C++ function), 51
cepton_sdk::util::FrameDetector (C++ class), 50
cepton_sdk::util::FrameDetector::FrameDetector (C++ function), 50
cepton_sdk::util::FrameDetector::get_options (C++ function), 50
cepton_sdk::util::FrameDetector::reset (C++ function), 50
cepton_sdk::util::FrameDetector::set_options (C++ function), 50
cepton_sdk::util::FrameDetector::update (C++ function), 50
cepton_sdk::util::get_timestamp_usec (C++ function), 48
cepton_sdk::util::OrganizedCloud (C++ class), 51
cepton_sdk::util::OrganizedCloud::CellInfo (C++ class), 52
cepton_sdk::util::OrganizedCloud::CellInfo::occupied (C++ member), 52
cepton_sdk::util::OrganizedCloud::CellInfo::original_index (C++ member), 52
cepton_sdk::util::OrganizedCloud::getIndex (C++ function), 51
cepton_sdk::util::OrganizedCloud::height (C++ member), 51
cepton_sdk::util::OrganizedCloud::info_cells (C++ member), 52
cepton_sdk::util::OrganizedCloud::n_returns (C++ member), 52
cepton_sdk::util::OrganizedCloud::points (C++ member), 52
cepton_sdk::util::OrganizedCloud::timestamp_end (C++ member), 51
cepton_sdk::util::OrganizedCloud::timestamp_start (C++ member), 51
cepton_sdk::util::OrganizedCloud::width (C++ member), 52
cepton_sdk::util::Organizer (C++ class), 52
cepton_sdk::util::Organizer::binSize (C++ function), 53
cepton_sdk::util::Organizer::CENTER (C++ enumerator), 52
cepton_sdk::util::Organizer::mode (C++ function), 53

cepton_sdk::util::Organizer::organize_points (C++ function), 52
cepton_sdk::util::Organizer::Organizer (C++ function), 52
cepton_sdk::util::Organizer::OrganizerMode (C++ enum), 52
cepton_sdk::util::Organizer::OrganizerSettings (C++ class), 53
cepton_sdk::util::Organizer::OrganizerSettings::enable_organizing (C++ member), 53
cepton_sdk::util::Organizer::OrganizerSettings::max_dt (C++ member), 53
cepton_sdk::util::Organizer::OrganizerSettings::mode (C++ member), 53
cepton_sdk::util::Organizer::OrganizerSettings::update_rate (C++ member), 53
cepton_sdk::util::Organizer::RECENT (C++ enumerator), 52
cepton_sdk::util::Organizer::settings (C++ function), 53
cepton_sdk::util::SensorPoint (C++ class), 48
cepton_sdk::util::SensorPoint::distance (C++ member), 48
cepton_sdk::util::SensorPoint::flags (C++ member), 49
cepton_sdk::util::SensorPoint::image_x (C++ member), 48
cepton_sdk::util::SensorPoint::image_z (C++ member), 48
cepton_sdk::util::SensorPoint::intensity (C++ member), 48
cepton_sdk::util::SensorPoint::return_type (C++ member), 48
cepton_sdk::util::SensorPoint::saturated (C++ member), 49
cepton_sdk::util::SensorPoint::timestamp (C++ member), 48
cepton_sdk::util::SensorPoint::valid (C++ member), 49
cepton_sdk::util::SensorPoint::x (C++ member), 49
cepton_sdk::util::SensorPoint::y (C++ member), 49
cepton_sdk::util::SensorPoint::z (C++ member), 49
cepton_sdk::util::SensorPoint::[anonymous] (C++ member), 49
cepton_sdk_capture_replay_close (C++ function), 43
cepton_sdk_capture_replay_get_enable_loop (C++ function), 44
cepton_sdk_capture_replay_get_filename (C++ function), 44
cepton_sdk_capture_replay_get_length (C++ function), 44
cepton_sdk_capture_replay_get_position (C++ function), 44
cepton_sdk_capture_replay_get_speed (C++ function), 44
cepton_sdk_capture_replay_get_start_time (C++ function), 44
cepton_sdk_capture_replay_get_end_time (C++ function), 44
cepton_sdk_capture_replay_is_open (C++ function), 43
cepton_sdk_capture_replay_is_running (C++ function), 44
cepton_sdk_capture_replay_open (C++ function), 43
cepton_sdk_capture_replay_pause (C++ function), 44
cepton_sdk_capture_replay_resume (C++ function), 44
cepton_sdk_capture_replay_resume_blocking (C++ function), 44
cepton_sdk_capture_replay_resume_blocking_once (C++ function), 44
cepton_sdk_capture_replay_seek (C++ function), 44
cepton_sdk_capture_replay_set_enable_loop (C++ function), 44
cepton_sdk_capture_replay_set_speed (C++ function), 44
cepton_sdk_clear (C++ function), 38
CEPTON_SDK_CONTROL_DISABLE_DISTANCE_CLIP (C++ enumerator), 36
CEPTON_SDK_CONTROL_DISABLE_IMAGE_CLIP (C++ enumerator), 36
CEPTON_SDK_CONTROL_DISABLE_NETWORK (C++ enumerator), 36
CEPTON_SDK_CONTROL_ENABLE_CROSSTALK_FILTER (C++ enumerator), 36
CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS (C++ enumerator), 36
CEPTON_SDK_CONTROL_ENABLE_STRAY_FILTER (C++ enumerator), 36
CEPTON_SDK_CONTROL_HOST_TIMESTAMPS (C++ enumerator), 36
cepton_sdk_create_frame_options (C++ function), 37
cepton_sdk_create_options (C++ function), 37
cepton_sdk_deinitialize (C++ function), 38
CEPTON_SDK_FRAME_COVER (C++ enumerator), 36
CEPTON_SDK_FRAME_CYCLE (C++ enumerator), 37
CEPTON_SDK_FRAME_MODE_MAX (C++ enumerator),

37
CEPTON_SDK_FRAME_STREAMING (C++ enumerator), 36
CEPTON_SDK_FRAME_TIMED (C++ enumerator), 36
cepton_sdk_get_control_flags (C++ function), 38
cepton_sdk_get_error (C++ function), 35
cepton_sdk_get_frame_length (C++ function), 39
cepton_sdk_get_frame_mode (C++ function), 39
cepton_sdk_get_n_sensors (C++ function), 41
cepton_sdk_get_port (C++ function), 38
cepton_sdk_get_sensor_handle_by_serial_number (C++ function), 41
cepton_sdk_get_sensor_information (C++ function), 42
cepton_sdk_get_sensor_information_by_index (C++ function), 41
cepton_sdk_get_version_major (C++ function), 38
cepton_sdk_get_version_minor (C++ function), 38
cepton_sdk_get_version_string (C++ function), 38
cepton_sdk_has_control_flag (C++ function), 38
cepton_sdk_initialize (C++ function), 38
cepton_sdk_listen_image_frames (C++ function), 43
cepton_sdk_listen_network_packet (C++ function), 45
cepton_sdk_listen_serial_lines (C++ function), 46
cepton_sdk_set_control_flags (C++ function), 38
cepton_sdk_set_frame_options (C++ function), 38
cepton_sdk_set_port (C++ function), 38
cepton_sdk_unlisten_image_frames (C++ function), 43
cepton_sdk_unlisten_network_packet (C++ function), 45
cepton_sdk_unlisten_serial_lines (C++ function), 46
CEPTON_SENSOR_MODEL_MAX (C++ enumerator), 39
CEPTON_SUCCESS (C++ enumerator), 34
CeptonSDKControl (C++ type), 36
CeptonSDKFrameMode (C++ type), 36
CeptonSDKFrameOptions (C++ class), 37
CeptonSDKFrameOptions::length (C++ member), 37
CeptonSDKFrameOptions::mode (C++ member), 37
CeptonSDKFrameOptions::signature (C++ member), 37
CeptonSDKOptions (C++ class), 37
CeptonSDKOptions::control_flags (C++ member), 37
CeptonSDKOptions::frame (C++ member), 37
CeptonSDKOptions::port (C++ member), 37
CeptonSDKOptions::signature (C++ member), 37
CeptonSensorErrorCode (C++ type), 34
CeptonSensorHandle (C++ type), 39
CeptonSensorImagePoint (C++ class), 42
CeptonSensorImagePoint::distance (C++ member), 42
CeptonSensorImagePoint::flags (C++ member), 42
CeptonSensorImagePoint::image_x (C++ member), 42
CeptonSensorImagePoint::image_z (C++ member), 42
CeptonSensorImagePoint::intensity (C++ member), 42
CeptonSensorImagePoint::reserved (C++ member), 43
CeptonSensorImagePoint::return_type (C++ member), 42
CeptonSensorImagePoint::saturated (C++ member), 42
CeptonSensorImagePoint::segment_id (C++ member), 43
CeptonSensorImagePoint::timestamp (C++ member), 42
CeptonSensorImagePoint::valid (C++ member), 42
CeptonSensorImagePoint::[anonymous] (C++ member), 43
CeptonSensorInformation (C++ class), 39
CeptonSensorInformation::firmware_version (C++ member), 40
CeptonSensorInformation::flags (C++ member), 40
CeptonSensorInformation::formal_firmware_version (C++ member), 40
CeptonSensorInformation::gps_ts_day (C++ member), 40
CeptonSensorInformation::gps_ts_hour (C++ member), 40
CeptonSensorInformation::gps_ts_min (C++ member), 40
CeptonSensorInformation::gps_ts_month (C++ member), 40
CeptonSensorInformation::gps_ts_sec (C++ member), 40
CeptonSensorInformation::gps_ts_year

(*C++ member*), 40
CeptonSensorInformation::handle (*C++ member*), 39
CeptonSensorInformation::is_calibrated (*C++ member*), 41
CeptonSensorInformation::is_mocked (*C++ member*), 41
CeptonSensorInformation::is_nmea_connected (*C++ member*), 41
CeptonSensorInformation::is_over_heated (*C++ member*), 41
CeptonSensorInformation::is_pps_connected (*C++ member*), 41
CeptonSensorInformation::is_ptp_connected (*C++ member*), 41
CeptonSensorInformation::is_sync_firing_enabled (*C++ member*), 41
CeptonSensorInformation::last_reported_age (*C++ member*), 40
CeptonSensorInformation::last_reported_humidity (*C++ member*), 40
CeptonSensorInformation::last_reported_temperature (*C++ member*), 40
CeptonSensorInformation::major (*C++ member*), 40
CeptonSensorInformation::measurement_period (*C++ member*), 40
CeptonSensorInformation::minor (*C++ member*), 40
CeptonSensorInformation::model (*C++ member*), 40
CeptonSensorInformation::model_name (*C++ member*), 40
CeptonSensorInformation::ptp_ts (*C++ member*), 40
CeptonSensorInformation::reserved (*C++ member*), 40
CeptonSensorInformation::return_count (*C++ member*), 40
CeptonSensorInformation::segment_count (*C++ member*), 40
CeptonSensorInformation::serial_number (*C++ member*), 39
CeptonSensorInformation::unused (*C++ member*), 40
CeptonSensorInformation::[anonymous] (*C++ member*), 41
CeptonSensorModel (*C++ type*), 39

F

FpCeptonNetworkReceiveCallback (*C++ type*), 45
FpCeptonSensorImageDataCallback (*C++ type*), 43
FpCeptonSerialReceiveCallback (*C++ type*), 46

H

HR80T_R2 (*C++ enumerator*), 39
HR80W (*C++ enumerator*), 39

S

SORA_P60 (*C++ enumerator*), 39
SORA_P61 (*C++ enumerator*), 39
SORA_P90 (*C++ enumerator*), 39

V

VISTA_860_GEN2 (*C++ enumerator*), 39
VISTA_H120 (*C++ enumerator*), 39
VISTA_P60 (*C++ enumerator*), 39
VISTA_P61 (*C++ enumerator*), 39
VISTA_P90 (*C++ enumerator*), 39
VISTA_X120 (*C++ enumerator*), 39
VISTA_X15 (*C++ enumerator*), 39